REMARKS

Claims 1-20 are still pending in this application. Reconsideration of the application is earnestly requested.

The Present Invention

As pointed out in the Background section, recent computer viruses are being written in an interpreted computer language (such as a scripting language) and are being distributed in source code. First-generation computer viruses were generally written in assembly language and distributed in binary code. Current technology for detecting computer viruses is based on byte code matching algorithms where matches of suspected code with a virus pattern file indicates the presence of a virus. This technique can detect exact matches of the virus code, but is ineffective with a polymorph of a scripting virus. As such, the present invention is directed toward generating a virus signature for a scripting virus and its polymorphs and for detecting a scripting virus and its polymorphs.

Chess et al.

The *Chess et al.* reference (*Chess*) is not relevant to the claimed invention because *Chess* only discusses first-generation computer viruses that are written in assembly language and distributed in binary code. For example, Figures 4 and 5 show only that the virus is written in byte codes and throughout the specification reference is made to "blocks" or "bytes" (for example, column 2, line 40; column 6, lines 42-45). Nowhere in the specification of *Chess* is there any mention of an interpreted language, a scripting language, or distribution of a computer virus in source code.

Further, *Chess* is not relevant to the claimed invention because *Chess* describes a technique for identifying a virus and its characteristics by comparing a virus-infected computer program with the original, uninfected program. The present invention does not involve comparing a virus-infected program with the original uninfected program. Because *Chess* is not relevant to the claimed invention, it is not surprising that many of the features required by the independent claims are not taught or suggested by *Chess*.

Claims 5-10

Claim 5 requires "receiving a portion of interpreted language source code containing a computer virus." *Chess* does not teach or suggest an interpreted language that contains a computer virus nor source code that contains a virus. That portion of *Chess* relied upon only discloses host and infected files written in binary code or bytes. There is no discussion of source code or of an interpreted language.

Claim 5 also requires "generating a language-independent representation of the computer virus." That portion of *Chess* relied upon does not disclose generating a representation of the computer virus that is independent of the language in which is written. In fact, Figure 4 simply shows manipulation of byte codes, the language in which viruses Sample 1 and Sample 2 are written. A byte code is not independent of the language used.

Claim 5 also requires "storing the language-independent representation of the computer virus as a virus signature." Because *Chess* does not disclose a language-independent representation, it likewise does not disclose storing that representation as a virus signature. In fact, *Chess* does not concern itself with generating in storing a virus signature for future use. Column 11, lines 55-57 mentions that a signature can be extracted for identification, but there is no teaching that this signature is stored for later use.

Claim 8 requires that "the language independent representation is a linearized string of key actions." The Office Action relies upon column 12 of *Chess* for this feature. But this portion of *Chess* is referring to Figure 4 that simply shows that a computer virus is separated into sections of byte codes, some being variable over samples, some being constant. A byte code is not a "key action" as described in the Specification of the instant application; nor is the string "linearized." As described in the Specification of the instant application, "linearized" refers to rearranging portions of the virus code such that they appear in the order in which they are executed; Figure 4 simply shows that the byte codes of the computer virus are left as is.

Claim 10 requires "parsing the portion of interpreted language source code into tokens." The computer virus code shown in Figure 4 of Chess is not interpreted language source code that is parsed into tokens.

Claims 15-20

Those features of claims 15-20 similar to the above features of claims 5-10 that have already been discussed are likewise not taught or suggested by *Chess* for the above reasons.

Further, claim 15 requires "extracting key actions," "linearizing key actions," "determining the set of minimum key actions," and "storing the set of minimum key actions as a virus signature." Respectfully, it is pointed out that columns 11 and 12 of *Chess* do not involve extracting, linearizing or determining key actions. Again, it is pointed out that Figure 4 of *Chess* simply discloses placing the virus code into sections and determining which sections remain constant over samples and which sections are variable. None of the above-cited steps of claim 15 are taught or suggested in *Chess*.

Chandnani et al.

Chandnani et al. (*Chandnani*) discloses a technique for detection of polymorphic script language viruses using lexical analysis. Figure 2 is useful for an understanding of the disclosure. In order to create virus detection data useful for later identification of a computer virus, samples of script language virus code are stored in sample store 56. The detection processor 52 then processes the sample code to create virus detection data that is stored in code detection database 57. Virus detection data includes token patterns for matching and CRC signature checking data. (Paragraphs 48-55.)

A great deal of effort is also spent creating language description data that describes a target script language. Script language processor 51 processes script language rules from rule base 54 to create script language description data that is stored in database 55. (Paragraphs 34-47.) In order to detect a computer virus in script language code a data stream representing that code is fed into detection engine 53 that uses the language description data 55 and the virus detection data 57 to determine if a virus is present. (Paragraphs 57-67.)

Important to an understanding of the distinction between *Chandnani* and the claimed invention is the nature of the data stream that is fed into detection engine 53. Lexical analysis is used to convert the data stream into a stream of tokens, but there is no other processing performed upon the tokens and no further representation of the tokens is produced. (Paragraphs 60-62.) For example, paragraph 60 explains that the virus detection process has only two stages: tokenizing the data stream and processing the tokens using the detection data. Further, the

tokens are, or represent, particular constructs in the scripting language being input and are therefore dependent upon that particular scripting language. As is known in the art of parsing, a token is an identifiable element in a scripting language.

Claims 1-4

Claim 1 is a method for identifying a computer virus in interpreted language source code. After a portion of the code is received, the second step of the claim requires "generating a language independent-representation of the portion of the interpreted language source code." The third and fourth steps also require the "language-independent representation" limitation. Use of a language-independent representation is beneficial because it can better detect polymorphs of scripting language viruses.

By contrast, *Chandnani* does not teach or suggest generating a language-independent representation of the interpreted language source code to be scanned for a virus. *Chandnani* does show in Figure 2 that an incoming data stream representing script code is fed into a detection engine 53 that has a lexical analyzer. (Paragraphs 57-62.) The lexical analyzer converts the data stream into a stream of tokens but no other processing is performed on the tokens. A stream of tokens is not a language independent representation of the source code. A stream of tokens is very much a language dependent stream of characters, language constructs, symbols, etc., that represent a particular scripting language.

Because the second step of claim 1 is not taught or suggested, it is requested that the rejection be withdrawn.

Claim 3 requires that "the virus signature is a language independent representation of an interpreted language source code computer virus." The data stream of tokens in *Chandnani* is compared to the virus detection data from database 57. But the virus detection data is not a language independent representation of source code. The virus detection data is a pattern of tokens or a CRC signature check (Paragraph 51), neither of which is a language independent representation. Further, because the virus detection data is compared directly to the stream of tokens (the tokens representing the particular script language used), the virus detection data must also be in the form of the particular script language being used (Paragraph 64). If the virus detection data were in a *language independent* form it would not be possible to match the stream of tokens which are in a *language dependent* form. The Office Action cites paragraph 55 as disclosing that the virus signature is in a language independent representation, but paragraph 55

only discloses that DFA data is used to transition from one stage in a pattern match operation to the next. The actual pattern match data (i.e., the virus signature) is still in a language dependent form.

Claim 4 requires that both the source code and the virus signature "are presented as a linearized string of key actions." As discussed above, *Chandnani* does with a not disclose key actions nor a linearized string.

Claims 11-14

Claim 11 requires "extracting selected key actions from the tokenized source code," "linearizing the key actions to generate and executing thread," and "comparing the executing thread with a virus signature of a known virus." As pointed out above in the overview of *Chandnani*, once the incoming data stream is converted into a stream of tokens there is no other processing performed upon the tokens and no further representation of the tokens is produced. Paragraphs 57-64 only disclose that the incoming data stream is converted into a stream of tokens by a lexical analyzer; no further conversion or processing is performed. Thus, the required steps of "extracting selected key actions," "linearizing the key actions," and "comparing the executing thread" are not taught or suggested by the *Chandnani*. The Office Action relies upon paragraph 20, but paragraph 20 only discloses that the incoming data stream (the script language) is converted into a stream of tokens. The Office Action also cites paragraph 64 but this paragraph only discloses that a pattern is matched against the token stream; there is no "executing thread" because *Chandnani* does not linearize any key actions let alone the generate key actions from the tokens in the first place.

Reconsideration of this application and issuance of a Notice of Allowance at an early date are respectfully requested. If the Examiner believes a telephone conference would in any way expedite prosecution, please do not hesitate to telephone the undersigned at (612) 252-3330.

Respectfully submitted,
BEYER WEAVER & THOMAS
Jonathan O. Scott
Registration No. 39,364

BEYER WEAVER & THOMAS, LLP
P.O. Box 778
Berkeley, CA 94704-0778
Telephone: (612) 252-3330
Facsimile: (612) 825-6304